C-Vorkurs, Tag 5, 15. April 2016

Jonas Cleve, Sebastian Faase, Thomas Tegethoff

Dos and Don'ts – Teil 2

Datentypen

Don't:

- `short`, `int`, `long` verwenden, wenn nicht nötig
  - es ist jedoch *immer* `int main()`

Datentypen

Don't:

- `short`, `int`, `long` verwenden, wenn nicht nötig
  - es ist jedoch *immer* `int main()`

Do:

- Datentypen aus `inttypes.h` verwenden
  - `int8_t`, `uint8_t`, `int16_t`, ...

printf

Don't:

- `printf(message)`
- `uint8_t x = 10; printf("%d", x)`

printf

Don't:

- `printf(message)`
- `uint8_t x = 10; printf("%d", x)`

Do:

- `printf("%s", message)`
- `uint8_t x = 10; printf("%"PRIu8, x)`

Fehlerbehandlung

Don't:

- Darauf vertrauen, dass nichts schlimmes passiert

Fehlerbehandlung

Don't:

- Darauf vertrauen, dass nichts schlimmes passiert

Do:

- Rückgabewerte von Funktionen überprüfen
  - ```
    char *p = malloc(10);
    if (p == NULL) {
        perror("Kein Speicher mehr!");
        exit(EXIT_FAILURE);
    }
    ```
- Manpage lesen um Fehler-Rückgabewerte herauszufinden
    ```

Strings und Arrays

Don't:

- String-Funktionen ohne `n` verwenden, die keine Längen prüfen
  - `strcpy` -> `strncpy`, `strcmp` -> `strncmp`, ...

Strings und Arrays

Don't:

- String-Funktionen ohne `n` verwenden, die keine Längen prüfen
  - `strcpy` -> `strncpy`, `strcmp` -> `strncmp`, ...

Do:

- Arrays nur von `0` bis `laenge-1` indizieren
- Das zusätzlich benötigte Null-Byte bei Strings beachten

Self-Check

Frage 01

- Welche Funktion muss immer existieren, damit ein Programm ausgeführt werden kann? Gib die vollständige Signatur an (für Parameterübergabe).

Frage 01

- Welche Funktion muss immer existieren, damit ein Programm ausgeführt werden kann? Gib die vollständige Signatur an (für Parameterübergabe).
- `int main(int argc, uint8_t *argv[])`

Frage 01

- Welche Funktion muss immer existieren, damit ein Programm ausgeführt werden kann? Gib die vollständige Signatur an (für Parameterübergabe).
- `int main(int argc, uint8_t *argv[])` $\Leftarrow$ Falsch!
- `int main(int argc, char **argv)`

Frage 02

- Die Variable `uint64_t zahl` wird im Programm verwendet.
  Was kann bei folgendem Ausdruck schiefgehen?
  `printf("%d\n", zahl)`
  Wie ist es korrekt?

Frage 02

- Die Variable `uint64_t zahl` wird im Programm verwendet.
  Was kann bei folgendem Ausdruck schiefgehen?
  `printf("%d\n", zahl)`
  Wie ist es korrekt?
- %d ist für Datentyp `int` gedacht. `uint64_t` kann deutlich größer werden.

Frage 02

- Die Variable `uint64_t zahl` wird im Programm verwendet. Was kann bei folgendem Ausdruck schiefgehen?
  `printf("%d\n", zahl)`
  Wie ist es korrekt?
- %d ist für Datentyp `int` gedacht. `uint64_t` kann deutlich größer werden. Weiterhin wurde das Semikolon vergessen!

Frage 02

- Die Variable `uint64_t zahl` wird im Programm verwendet. Was kann bei folgendem Ausdruck schiefgehen?
  `printf("%d\n", zahl)`
  Wie ist es korrekt?
- %d ist für Datentyp `int` gedacht. `uint64_t` kann deutlich größer werden. Weiterhin wurde das Semikolon vergessen!
- `printf("%"PRIu64"\n", zahl);`

Frage 03

- Was ist die Ausgabe von folgendem Programmauszug?

```
uint8_t i = 20;
for (uint8_t i = 0; i<3; i++) {
    uint8_t i = 5;
    printf("%"PRIu8"\n", i);
}
printf("%"PRIu8"\n", i);
```

Frage 03

- Was ist die Ausgabe von folgendem Programmauszug?

```
uint8_t i = 20;
for (uint8_t i = 0; i<3; i++) {
    uint8_t i = 5;
    printf("%"PRIu8"\n", i);
}
printf("%"PRIu8"\n", i);
```

- 5
  5
  5
  20

Frage 04

- Was ist die Ausgabe von folgendem Programmauszug?

```
char *buffer = "Hallo";
char term = 'Z';
char *pTerm = &term;

buffer = pTerm;

printf("%s\n", buffer);
```

Frage 04

- Was ist die Ausgabe von folgendem Programmauszug?

```
char *buffer = "Hallo";
char term = 'Z';
char *pTerm = &term;

buffer = pTerm;

printf("%s\n", buffer);
```

- z.B.

```
Z?°¾Íÿ
ZÏ@÷wü
ZÃý
```

Feedback